vordenker
Summer-Edition 2017

— vordenker-archive —

Rudolf Kaehr

(1942-2016)

Title

Introducing and Modelling Polycontextural Logic

Archive-Number / Categories

1_31 / K07

Publication Date

1996

Keywords

Combinatory Logic, Computational Reflection, Functional Programming, Lambda-calculus, Parallel Processing, Polycontexturality, Proemial Relationship, Semiotics

Disciplines

Artificial Intelligence and Robotics, Logic and Foundations, Theory of Science, Cybernetics

Abstract

Gotthard Günther introduced the proemial-relationship (PRS) as one of the basic trans-classical concepts of polycontexturality. PRS pre-faces and constitutes as the mechanism of the difference making 'difference' all relational and operational orders. The present paper develops a first step modelisation of the proemial-relationship in analogy to graph-reduction based implementations of functional languages. A proemial-combinator, PR, is designed and implemented, which is proposed as an extension of functional programming languages and as an implementation technique for process-communication and computational reflection.

Citation Information / How to cite

Rudolf Kaehr & **Thomas Mahler**: "Introducing and Modeling Polycontextural Logics", www.vordenker.de (Sommer Edition, 2017) J. Paul (Ed.),

URL: http://www.vordenker.de/rk/rk-tm_Introducing-and-Modelling-Polycontextural-Logic_1996.pdf – originally presented at: Thirteenth European Meeting on Cybernetics and Systems Research 1996 - EMCSR Wien 1996, April 9-12

Categories of the RK-Archive

K01	Gotthard Günther Studies	K08	Formal Systems in Polycontextural Constellations
K02	Scientific Essays	K09	Morphogrammatics
K03	Polycontexturality – Second-Order-Cybernetics	K10	The Chinese Challenge or A Challenge for China
K04	Diamond Theory	K11	Memristics Memristors Computation
K05	Interactivity	K12	Cellular Automata
K06	Diamond Strategies	K13	RK and friends
K07	Contextural Programming Paradigm		

Introducing and Modeling Polycontextural Logics

R. Kaehr and **Th. Mahler**
Institut für Kybernetik und Systemtheorie – ICS
Am Hülsenbusch 54, D-44803 Bochum
E-Mail: ics@ics.prima.ruhr.de

Abstract

Gotthard Günther introduced the proemial-relationship (PRS) as one of the basic trans-classical concepts of polycontexturality. PRS pre-faces and constitutes as the mechanism of the difference making 'difference' all relational and operational orders. The present paper developes a first step modelisation of the proemial-relationship in analogy to graph-reduction based implementations of functional languages. A proemial-combinator, PR, is designed and implemented, which is proposed as an extension of functional programming languages and as an implementation technique for process-communication and computational reflection.

Keywords: combinatory logic, computational reflection, functional programming, lambda-calculus, parallel processing, polycontexturality, proemial relationship, semiotics.

1 Introducing and Modeling Polycontextural Logics

The idea of an extension of classical logic to cover simultaneously active ontological locations was introduced by Gotthard Günther (1900-1984, us-american thinker, born in germany; colleague of Heinz von Foerster at the BCL, Urbana). The ideas of Polycontextural Logic originate from Günthers study of Hegel, Schelling and the foundation of cybernetics in cooperation with Warren St. McCulloch [Gun95]. His aim was to develop a philosophical theory and mathematics of dialectics and self-refential systems, a cybernetic theory of subjectivity as an interplay of cognition and volition.

Polycontextural Logic is a *many-system* logic, a dissemination of logics, in which the classical logic systems (called contextures) are enabled to interplay with each other, resulting in a complexity which is structuraly different from the sum of its components [Kae81] [Pfa91]. Although introduced historicaly as an interpretation of many valued logics, polycontextural logic does not fall into the category of fuzzy or continous logics or other deviant logics. Polycontextural logics offers new formal concepts such as multi-negational and transjunctional operators.

The world has infinitely many logical places, and it is representable by a two-valued system of logic in each of the places, when viewed isolately. However, a coexistence, a heterarchy of such places can only be described by the proemial relationship in a polycontextural logical system. We shall call this relation according to Günther the proemial relationship, for it prefaces the difference between relator and relatum of any relationship as such. Thus the proemial relationship provides a deeper foundation of logic and mathematics as an abstract potential from which the classic relations and operations emerge.

The proemial relationship rules the mechanism of distribution and mediation of formal systems (logics and arithmetics), as developed by the theory of polycontexturality. This relationship was characterised as the simultaneous interdependence of order and exchange relations between objects of different logical levels.

According to Günther ([Gun80b], p. 226): *The proemial relationship belongs to the level of the kenogrammatic structure because it is a mere potential which will become an actual relation only as either symmetrical exchange relation or non-symmetrical ordered relation. It has one thing in common with the classic symmetrical exchange relation, namely, what is a relator may become a relatum and what was a relatum may become a relator. Or to put it differently: what was a distinction may become something which is distinguished, and what has been distinguished may become a process of distinction. The proemial relationship crosses the distinction between form and matter. [...] We can either say that proemiality is an exchange founded on order; but since the order is only constituted by the fact that the exchange either transports a relator (as relatum) to a context of higher logical complexities or demotes a relatum to a lower level, we can also define proemiality as an ordered relation on the base of an exchange.*

The proemial relationship implies the simultaneous distribution of the same object over several logical levels, which is not covered by classical theories of types. In the following, a concept of such a coexistence and parallelism will be developed which models the kenogrammatic proemial relationship ([Gun80a], [Gun80b]).

Due to the special properties of the proemial relationship and the limitations of classical calculi, an algebraic representation of the proemial relationship must be self-referential, i.e. in classical formalisms it has a paradoxical and antinomic structure. Because of these fundamental difficulties with its formalisation, an attempt will be made here to develop an *operational* model of the proemial relationship.

To do this, the operational semantics of an abstract combinatorical machine will be extended by a proemial combinator PR [Dav92].

The proemial relationship describes the interdependence of an order relation $\longrightarrow$, and an exchange relation $\Updownarrow$, between two operators and operands respectively. Between the operands x_i and x_{i-1} and the operators R_{i+1} and R_i exists a categorical coincidence relationship, which constitutes the simultaneity of the distributed order and exchange relations. Between the constituents of the proemial relationship (order, exchange, coincidence and locations) there exists a chiastic interlocking mechanism of mutual foundations. Thus the concept of proemiality is not a concept of the logic of relations (Peirce, Schröder) but prefaces – like the *differance* (Derrida) – all concepts of relations as such [Kae95].

$$PR\Big(R_{i+1}, R_i, x_i, x_{i-1}\Big) :=$$

$$\begin{tikzcd}
R_{i+1} \arrow[r] & x_i \arrow[d, Leftrightarrow] & \\
 & R_i \arrow[r] & x_{i-1}.
\end{tikzcd}$$

We distinguish the *open* from the *closed* form of the proemial relationship ([Kae78], p. 5 f.). The closed proemial relationship is cyclical:

$$\begin{tikzcd}
R_{i+1} \arrow[r] \arrow[d, Leftrightarrow] & x_i \arrow[d, Leftrightarrow] \\
x_{i-1} & R_i \arrow[l]
\end{tikzcd}$$

i.e. $PR(PR^i) = PR^i$. A different relationship holds for the open proemial relationship: $PR(PR^i) = PR^{(i+1)}$. It has the following form:

$$\begin{tikzcd}
i+1\colon & R_{i+2} \arrow[r] & x_{i+1} \arrow[d, Leftrightarrow] & & \\
i\colon & & R_{i+1} \arrow[r] & x_i \arrow[d, Leftrightarrow] & \\
i-1\colon & & & R_i \arrow[r] & x_{i-1}
\end{tikzcd}$$

However, the simultaneous distribution of the *same* λ-Terms over several logical levels cannot be modelled in the λ-Calculus ([Cur69], [Bar80]).

The λ-Term $(f\ x)(x\ f)$ uses f and x inside an expression both as an operator and as an operand. Inside the λ-Calculus itself, as in all semiotically grounded calculi, the *identity* of terms can be expressed as unique objects. However their *sameness* cannot be expressed.

The semiotic equality of the symbol `$f$' in `$(f\ x)$' with the symbol `$f$' in `$(x\ f)$' says nothing about whether, within a reduction of the λ-Terms, these identical terms are also physically equal, i.e. handled as the *same* object.

The reason for this lies in the Token–Type–Relation of the Semiotics on which the λ-Calculus is grounded. The Token–Type–Relation subsumes all physically different, but *equally represented* Tokens under one Type.

Since this step involves an abstraction from the physical realisation of the tokens, it is not possible to infer the physical identity or non-identity of different occurrences of symbol sequences from the terms of a semiotically grounded calculus.

A calculus only operates with types and since the type equality (or equivalently form equality $\equiv_{sem}$) does not also imply token equality (i.e. physical or pointer equality $\equiv_z$), it follows that such a calculus cannot represent any concept of sameness.

Moreover, from the syntactic structure of λ terms, one cannot deduce which reduction procedure should be used to evaluate them. The above expression does not indicate whether $(f\ x)$ and $(x\ f)$ should be evaluated sequentially (and if so, in what order) or simultaneously.

The question of the sameness of λ-term representations along with the choice of evaluation strategy remains solely a matter of implementation techniques for the λ-Calculus, since it does not concern the (semiotically grounded) term semantics.

In contrast, the proemial relationship introduces the idea of a simultaneous distribution of the same object over several reference systems which is exactly what was not representable in the λ-Calculus. The model of the proemial relationship proposed here is an attempt to represent exactly this behaviour. It is based on Günther's conception of kenograms as empty slots, where semiotic processes can be inserted.

The sameness of a term (which cannot be defined on the semiotic level) is then determined through the sameness of the kenogram where it is inserted. This term which has been realised in one and the same kenogram, can now serve simultaneously as operator and operand within different semiotic processes[1].

1.1 Implementation of the Proemial Combinator PR

Based on the fundamental idea of the sameness of semiotic processes within kenogrammatics, the operational semantics of the proemial combinator $PR\big(R_{i+1}, R_i, x_i, x_{i-1}\big)$ can now be determined by means of the operational semantics of a virtual combinator machine ([Tur79], [Dil88], [Dav92]).

This model makes use of the homogeneity of programs and data of the graph representation for the combinator machine. In this way, a certain node z, which is realised as a physical object at a particular store address, can serve both as an operator and as an operand within different application nodes.

Due to the parallel architecture of the combinator machine, this exchange of roles (Operator $\Longleftrightarrow$ Ope-

[1]For the relationship between semiotics and kenogrammatics, see [Gun80c], [Kae82], [Kae92], [Kae95], [Mah93]

rand) within the same node z can be executed simultaneously.

R_i, R_{i+1}, x_i and x_{i-1} can be arbitrary nodes of the combinator graph. The order relation of the proemial relationship, $\longrightarrow$, represents here the application `app(rator,rand)`, which always guarantees a unique distinction between operator and operand:

$$\begin{tikzcd}
 & @ \arrow[dl] \arrow[dr] & & & @ \arrow[dl] \arrow[dr] & \\
R_{i+1} & & x_i & R_i & & x_{i-1}
\end{tikzcd}$$

Two such application nodes can occur in arbitrary positions within a combinator program. Within such a node, the role of a subnode as operator or operand is always determined through the application structure.

If pointers are used in the combinator graph, it is possible that x_i and R_i indicate the *same* physical Object z. Inside of $\mathtt{app}(R_{i+1}, x_i)$, z plays the role of operand and in $\mathtt{app}(R_i, x_{i-1})$ it plays the role of operator:

$$\begin{tikzcd}
 & @ \arrow[dl] \arrow[dr] & & @ \arrow[dl] \arrow[dr] & \\
R_{i+1} & & \boxed{x_i \equiv_z R_i} & & x_{i-1}
\end{tikzcd}$$

The pointer equality of x_i und R_i, $x_i \equiv_z R_i$, has the effect that z plays the role of operator and operand within different applications (i.e. order relationships). This position exchange within the order relation `app(rator,rand)` serves as a model for the exchange relation of the proemial relationship, $\Leftrightarrow$. In this way, R_i, R_{i+1}, x_i and x_{i-1}, with $R_i \equiv_z x_i$ satisfies the schema for the proemial relationship:

$$\begin{tikzcd}
R_{i+1} \arrow[r] & x_i \arrow[d, Leftrightarrow] & \\
 & R_i \arrow[r] & x_{i-1}.
\end{tikzcd}$$

The simultaneity of logical levels required by the informal specification of the proemial relationship, where z occupies different positions of the order relation `app(rand,rator)`, will now be modelled so that the application $\mathtt{app}(R_{i+1}, x_i)$ and $\mathtt{app}(R_i, x_{i-1})$ can be simultaneously evaluated.

The physical object z then serves simultaneously (in the sense of the parallel architecture used) as operator and operand within different applications. This theoretical concept leads to the following implementaion of the proemial combinator PR:

```
|apply (PR, (stack as ( ref(app((_,R1),_,_))::
                        ref(app((_,R2),_,_))::
                        ref(app((_,x1),_,_))::
                        (node as
                (ref(app((_,x2),_,_)))))::_))) =
let
 val first =  ref(app((R1,x1),ref Eval,ref []));
 val second = ref(app((R2,x2),ref Eval,ref []));
in
 (node :=
  app((ref(app((ref(comb(CONS,ref Ready,ref [])),
                first),ref Ready,ref [])),
       second),ref Ready,ref []);
  parEval (last stack, first);
  parEval (last stack, second))
end
```

```
fun parEval (root,node) =
 let

  val emark = get_mark node;
  val wq = get_q node;
 in
  if (! emark = Ready) then ()
  else if (! emark = Busy) then
   (make_wait root;
    wq := root::(! wq))
  else
   (emark := Busy;
    newTask node)
  end;
```

The reduction of the combinator `PR` expects four arguments, `R1, R2, x1` and `x2`. From these, two applications `first = (app(R1,x1))` and `second = (app(R2,x2))` are constructed.

Both these nodes are first combined into one cons object `CONS(first,second)`. In this way some results of the applications `first` and `second` remain and can be inspected later.

In the next step, both the applications are given over to the scheduling mechanism as parallel processes via `(parEval first)` and `(parEval second)`.

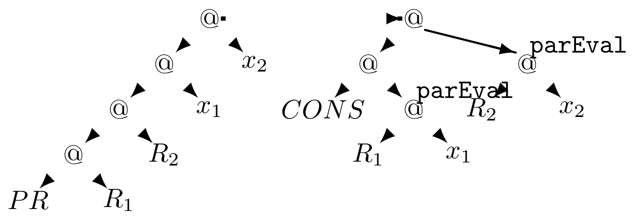


figure 1: The graph reduction of $PR(R_1, R_2, x_1, x_2)$

This reduction schema is represented in Fig.(1). The marking `parEval` on the nodes $@(R_1, x_1)$ and $@(R_2, x_2)$ indicates that the reduction does not run strictly but neither does it run non-strictly: the nodes are not evaluated until they are combined into a $CONS$ object.

The synchronisation by means of `(make_wait root; wq := [root])` can be ommitted, since `first` and `second` are not subprocesses of an overruling strict operator which waits for their computation results. Instead they are autonomous, proemially connected processes.

If R_2 and x_1 are realized as the *same* physical object z then clearly the following reduction schema results (figure 2).

In the context of R_1, z is an operand but in the context of x_2, an operator. Since both applications $@(R_1, z)$ and $@(z, x_2)$ are evaluated in parallel, this reduction schema satisfies the exchange relation of the proemial relationship.

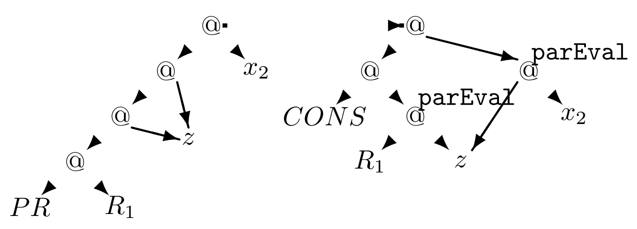


figure 2: The graph reduction of $PR(R_1, z, z, x_2)$

$$\begin{tikzcd}
\text{metacomp.:} & PRG_2 \arrow[r] & DAT_1 \arrow[d, Leftrightarrow] & \\
\text{objectcomp.:} & & PRG_1 \arrow[r] & DAT_0
\end{tikzcd}$$

$\longrightarrow$: program PRG_i is applied to data DAT_{i-1}.
$\Updownarrow$: PRG_i becomes DAT_i redarding the metacomputation PRG_{i+1} and vice versa.

figure 3: Locigal levels of Computational Reflection

1.2 Application Possibilities

Meta-level Architectures and Reflective Programing

Under the key words *Computational Reflection* (CR) and *Meta-level-Architectures* in fundamental computing research, attempts are made to extend the classical concept of computation, for example as it is formulated in the λ-Calculus. In particular, the problem concerns the development of computation systems which `reflect' over their computations.

According to Maes ([Mae88], p.22 ff.; [Smi82]) a reflective programming language has the property that it explicitly makes methods available for reflective computation.

In concrete this means that:

1. The interpreter for such a language must ensure that every program to be evaluated has access to the data structures representing the program itself (or certain aspects of it). Such a program then has the possibility of manipulating the data containing its own representation (meta-computation).
2. The interpreter must further guarantee that a (*causal connection*) exists between these data and the aspects of the program which represent them. The modification which a reflective program carries out on its representation also modifies the state and further execution of the program (object computation). In this sense, the meta-computation is reflected in the object computation.

In such a system, representations of computation instructions can either be evaluated *as a program* on the object computation level, or alternatively (for example, in an error situation) they can serve *as the data* of a meta-computation level which could, for example, correct the error.

This structure is represented in the following figure (3). The exchange between the operator PRG_i of the object computation and the operand DAT_i of the meta-computation can be described by the exchange relation of the proemial relationship $\Updownarrow$.

The distinction between the program (operator) and data (operand) within the one computation level corresponds to the order relation of the proemial relationship.

It follows that the structure schema of a reflective computation system corresponds exactly to that of the proemial relationship, which is not the same as *(Eigen)-behavior* [Foe76]. Eigen-values and fixpoint semantics needs transfinite recursions and are only of value for describtions and not for finite constructions of artificial systems.

Since the proemial combinator $PR\ x_i$ and R_i exists as a single physical object, all modifications to x_i also act directly on R_i. In this way the pointer equality $x_i \equiv_z R_i$ guarantees the causal connection between meta-level and object level.

The proemial combinator PR is therefore suitable for the modelling of reflective systems in the sense of Maes's definition.

In existing reflective systems (e.g. 3LISP[Smi82]) the meta and object levels are not realised as simultaneous processes, but instead execute purely sequentially.

It follows that the meta-computations PRG_{i+1} which manipulate the object programs's representation as data DAT_i, do not simultaneously influence the object computation ($PRG_i \longrightarrow DAT_{i-1}$).

They become effective only when the meta-computation is terminated and the object level is again activated, i.e. when the modified instructions for the object computation are given over to the interpreter.

This means that, at any particular point in time, the whole computation will either be evaluated on the meta-level or on the object level. It is always uniquely determined whether an instruction serves as program (operator) or as data (operand).

In contrast, the proemial combinator PR, along with the programs based on it, enables the *simultaneous coupling* of object- and meta-computation.

In this way PR offers a parallel modelling concept for reflective systems.

Generalisation of the Concept of Parallelism

The definition of the proemial combinator PR is based on the physical coupling of parallel computations. This modelling approach will now be extended to a kenogrammatic notation for parallel processes.

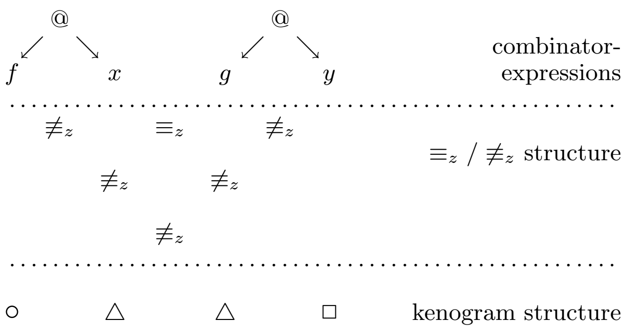
To do this, the structure of parallel processes in the above model will first be examined.

Kenogrammatics describes a pre-semiotic domain in which the law of graphemic identity does not govern. Kenogrammatics relates to polycontextural systems as formal semiotics to classical calculi and embraces semiotics itself.

A computation $PR\ f\ g\ x\ y$ has the form of the open proemial relationship if $g \equiv_z x$ and $f \not\equiv_z y$. If $g \equiv_z x$

and $f \equiv_z y$ then the computation corresponds to the closed form.

The type of proemial relationship which applies to the parallel evaluation of two combinator expressions $(f\ x)$ and $(g\ y)$ cannot be determined from the term structure; instead it can only be found out by looking at the structure of pointer equality $(\equiv_z)$ and pointer difference $(\not\equiv_z)$ of f, x, g and y.



The $\equiv_z$ / $\not\equiv_z$-structure ist structure-isomorphic to the ϵ/ν-structure of kenogram complexes [Mah93].

Due to this isomorphism, the above $\equiv_z$ / $\not\equiv_z$–structure of f, x, g, y can be represented as the kenogram sequence ○△△□ .

Since here $x \equiv_z g$ and $f \not\equiv_z y$, the sequence ○△△□ corresponds to the open proemial relationship.

The kenogram sequence ○△△○ indicates the closed proemial relation, since here $x \equiv_z g$ and $f \equiv_z y$. The kenogram sequence ○○○○ represents a situation in which all four arguments of the proemial relationship refer to the same object.

In addition, the sequences ○○○△ and ○△△△ indicate proemial relationships, since $x \equiv_z g$ applies to them also. However, the remaining ten of the total of fifteen kenogram sequences of length 4 indicate situations of pointer equality and difference which do not satisfy the proemial relationship (since $x \not\equiv_z g$ there is no exchange relation).

Since proemial conditions only occur in some of the total number of combinations for kenogram sequences, it is possible to regard the proemial relationship as a special case of a more general concept of parallelism which allows all possible $\equiv_z$ / $\not\equiv_z$-structures.

Due to the isomorphism between $\equiv_z$ / $\not\equiv_z$- and ϵ/ν-structures, the structure of parallel computation enabled by the PR combinator can be described within the framework of kenogrammatics.

Physical coupling and interaction between processes can then generally be formally represented by kenogrammatic operations.

For example, if a computation $(h\ z)$ with structure ○△ is to be added to two existing parallel computations $(f\ x)$ and $(g\ y)$ having a (closed proemial) structure ○△△○ , then for the resulting combined computation $(f\ x)\|(g\ y)\|(h\ z)$ several ϵ/ν–structures are possible:

$$\begin{array}{ccccccccccc}
f & & x & & g & & y & & h & & z \\
\hline
 & \nu & & \epsilon & & \nu & & ?_1 & & \nu & \\
 & & \nu & & \nu & & ?_2 & & ?_5 & & \\
 & & & \nu & & ?_3 & & ?_6 & & & \\
 & & & & ?_4 & & ?_7 & & & & \\
 & & & & & ?_8 & & & & &
\end{array}$$

The resulting eight degrees of freedom $?_i$ can be filled with ϵ or ν [Mah93]. The possible kenogram sequences are then:

$$\begin{array}{rcl}
○△△○ \ @ \ ○△ & = & \{ ○△△○○△,\ ○△△○○□, \\
 & & ○△△○△○,\ ○△△○△□, \\
 & & ○△△○□○,\ ○△△○□△, \\
 & & ○△△○□\star\}.
\end{array}$$

The indexed chain $@_i$ determines single elements of this set. It can also be used to determine precisely specified couplings of parallel processes.

1.3 Prospects

In the model developed here, the transclassical aspects of the proemial relationship occur only (as shown) from the perspective of a particular interpretation of the proemial combinator PR as an emergent surface phenomenon. It does not belong to the architecture of the combinator machine as an inherent feature.

It may be said therefore, that the approach given here is not a transclassical model, but instead only a particular application and interpretation of a classical formalism.

This restriction must necessarily apply, since the model is formulated within the linguistic framework of classical formal systems and programming languages (ML, HASKELL) i.e. *positive languages.* In positive languages, the designational values of propositions are positive, in negative languages, the designational values are both positiv and negative.

A possible next step would to develop a new complete programming language for the computation model or to integrate it within existing systems. These programming languages could then be used for the implementation of coupled parallelism (in particular polycontextural logics and arithmetics, self-referential, heterarchical and autopoietic systems) [Kae88].

By means of such functional languages which would only require a few extended constructs, it would be possible to develop formal models of process communication and interaction of structurally complex systems (e.g. operating systems and artificial living systems).

The fundamental barrier to the representation of the proemial relationship lies in the concepts of object, symbol and identity in classical semiotics and all positive linguistic symbolic systems based on it (formal, algorithmic and autological systems).

Of particular interest for an adequate formalisation of the proemial relationship is therefore a critique and renewed conception of the model presented here from the perspective of Günther's Theory of *Negative Languages* [Gun79].

References

[Bar80] Barendregt, H.P.: *The Lambda-calculus. Its Syntax and Semantics.* Amsterdam, North-Holland, 1980.

[Cur69] Curry, H.B., Feys, R.: *Combinatory Logic.* Amsterdam, North-Holland, 1969.

[Dav92] Davies, A.J.T.: *An Introduction to Functional Programming Systems Using Haskell.* Cambridge Computer Science Texts 27, Cambridge University Press, 1992.

[Dil88] Diller, A.: *Compiling Functional Languages.* New York, John Wiley & Sons, 1988.

[Foe76] Von Foerster, H.: *Objects: Tokens for (Eigen)-behaviors.* In: ASC Cybernetics Forum VIII (3,4), S. 91-96, 1976.

[Gun78] Günther, G.: *Idee und Grundriß einer nicht-aristotelischen Logik. Die Idee und ihre philosophischen Voraussetzungen.* 2. Auflage, Hamburg, Verlag Felix Meiner, 1978.

[Gun79] Günther, G.: *Identität, Gegenidentität und Negativsprache.* In: *Hegeljahrbuch 1979*, Pahl-Rugenstein, pp.22-28.

[Gun80a] Günther, G.: *Beiträge zur Grundlegung einer operationsfähigen Dialektik.* Bd. 1-3, Hamburg, Verlag Felix Meiner, 1976-1980.

[Gun80b] Günther, G.: *Cognition and Volition. A Contribution to a Cybernetic Theory of Subjectivity.* In: [Gun80a] Bd.2.

[Gun80c] Günther, G.: *Natural Numbers in Trans-Classic Systems. Part I, II.* In: [Gun80a] Bd.2.

[Gun95] Günther, G.: *Number and Logos. Unforgettable hours with Warren St. McCulloch.* In: *Jahrbuch für Selbstorganisation Band 5: Realitäten und Rationalitäten.* Kaehr, R., Ziemke, A. (Eds.), Berlin, Huncker & Dumblot, 1995

[Har91] Van Harmelen, Frank: *Meta-level Inference Systems.* London, Pitman, 1991.

[Kae78] Kaehr, R.: *Materialien zur Formalisierung der dialektischen Logik und der Morphogrammatik 1973-1975.* In: [Gun78], Anhang.

[Kae81] Kaehr, R.: *Das graphematische Problem einer Formalisierung der transklassischen Logik.* In: Beyer, W.R.(Ed.): *Die Logik des Wissens und das Problem der Erziehung.* Hamburg, Felix Meiner Verlag, 1981

[Kae82] Kaehr, R.: *Einschreiben in Zukunft.* In: Hombach, D.(Ed.): *Zeta 01. Zukunft als Gegenwart.* Berlin, Verlag Rotation, 1982.

[Kae88] Kaehr, R., Goldammer, von E.: *Again Computers and the Brain.* In: *Journal of Molecular Electronics.* Vol. 4, 1988, p. 31-37

[Kae92] Kaehr, R.: *Disseminatorik: Zur Logik der 'Second Order Cybernetics'. Von den 'Laws of Form' zur Logik derReflexionsform.* In: *Kalkül der Form.*, Baecker, D. (Ed.), Frankfurt a. M., Suhrkamp Verlag, 1993.

[Kae95] Kaehr, R., Mahler, Th.: *Proömik und Disseminatorik. I. Abbreviaturen transklassischen Denkens, II. Operationale Modellierung der Proemialrelation.* In: *Jahrbuch für Selbstorganisation Bd.5: Realitäten und Rationalitäten.* Kaehr, R., Ziemke, A. (Eds.), Berlin, Huncker & Dumblot, 1995

[Mae88] Maes, P., Nardi, D.: *Meta-Level Architectures and Reflection.* Amsterdam, North–Holland, 1988.

[Mah93] Mahler, Th., Kaehr, R.: *Morphogrammatik. Eine Einführung in die Theorie der Form.* Klagenfurter Beiträge, 1994.

[Pfa91] Pfalzgraf, J.: *Logical Fiberings and polycontextural systems.* In: *Fundamentals of Artificial Intelligence Research*, Ph. Jorrand, J. Kelemen (Eds.), pp. 170-184, New York, Springer, 1991

[Smi82] Smith, B.C.: *Reflection and Semantics in a Procedural Language.* LCS Technical Report TR–272, MIT, Massachusets, 1982.

[Tur79] Turner, D.A.: *A New Implementation Technique for Applicative Languages.* Software Practise and Experience Bd. 9, 1979.